NASA CONTRACTOR REPORT CR177427

Dynamic Modelling and Estimation of the Error Due to Asynchronism in a Redundant Asynchronous Multiprocessor System

Loc c. Huynh and R. W. Duval

P.O. A30146C (HMK)
May 1986

NASA

NASA CONTRACTOR REPORT CR177427

Dynamic Modelling and Estimation of the Error Due to Asynchronism in a Redundant Asynchronous Multiprocessor System

Loc c. Huynh and R. W. Duval
Advanced Rotocraft Technology, Inc.
Mountain View, California
May 1986

Prepared for
Ames Research Center
Under P.O. A30146C (HMK)

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The use of Redundant Asynchronous Multiprocessor System to achieve Ultra reliable Fault Tolerant Control Systems shows great promise. The development has been hampered by the inability to determine whether differences in the outputs of redundant CPU's are due to failures or to accrued error built up by slight differences in CPU clock intervals. This study derives an analytical dynamic model of the difference between redundant CPU's due to differences in their clock intervals and uses this model with on-line parameter identification to identify the differences in the clock intervals. The ability of this methodology to accurately track errors due to asynchronisity is demonstrated using a simulated multiprocessor system. The algorithms generate an error signal with the effect of asynchronisity removed and this signal may be used to detect and isolate actual system failures.

I. BACKGROUND

Increased reliance on computers and distributed processing in flight control systems requires higher reliability. A new approach to achieving higher reliability is to distribute sequential computational operations over multiple Microprcessor Central Processing Units (CPU's.) Increased reliability of the sequential CPU's is then achieved using parallel pipelines of sequential CPU's, with each CPU comparing the results of the parallel operations at the previous sequential stage to identify and isolate failures. Such a Redundant Asynchronous Multiprocessor System (RAMPS) has been designed and tested [1]. Advantages of this system include simplified fault detection and isolation logic since the results at each sequential computation are generated redundantly for comparison. Also the reliability is higher than other redundant architectures since the operation of each CPU is independent of failures in any other CPU.

Using totally autonomous CPU's to achieve high reliability results in a major impediment to successful implementation of this system. Since each CPU must have its own clock to be totally independent of other CPU's, the parallel computations will be performed asynchronously. Small differences in the CPU clocks can propagate into large differences in the parallel computations over a period of time and the fault detection logic will be unable to distinguish between errors due to CPU failures and errors due to asynchronisity.

One approach to resolving the problem of asynchronisity has been to use asymptotically stable control laws in each CPU [2]. This assumes that the error due to asynchronisity will be bounded. The disadvantage of this approach is that unrealistic constraints on the control laws may be required to reduce the error due to asynchronisity to acceptable levels.

II. OBJECTIVE AND OUTLINE OF REPORT

The objective of this study is to examine the nature of the error due to asynchronisity to determined if it is possible to model and track this source of error. If this is possible, then this information can be used by the fault detection logic to account for differences in parallel CPU's due to asynchonisity and it can also be used to periodically resynchronize the CPU's to prevent excessive build up of this error source.

This report begins with a description of the approach used in accomplishing this objective. A description of the RAMP system and the simulation of this system on a single CPU computer then follows. The next section defines the error due to asynchronisity and examines the nature of this error using the computer simulation of the RAMP system. A mathematical model of this error is then developed and tested against the error generated by the simulation. Finally, on-line parameter identification is applied, using the developed error model, to identify the difference in the CPU clocks and use this information to estimate the error due to asynchronisity on-line. The algorithms are teste using the simulated RAMP system.

III. APPROACH

In order to efficiently attack the problem, a simulation program was implemented to study the behavior of RAMPS. The approach now can be presented in the following order:

1. Determine operational requirements to insure analytical error function: The definition of the error function affects the nature of the error. The difference in the output of two assynchronous CPU's is a highly nonanalytic function, however by "synchronizing" the time at which this error is observed by a downstream CPU, the observed error function can be made smooth and hence analytically describable.

2. Derive a mathematical model of the analytical error as function of the difference in CPU clock intervals:The error between parallel asynchronous CPU's will be defined and the analytical error function will be derived by applying Sample Data theory.

3. Identify the difference in CPU clocks by applying regression to error data with the error model as a constraint.

4. Use the identified difference in the CPU clocks to track the error due to asynchronism. The voting logic can then be based on other sources of error. Two voting logics will be compared here:

- Midvalue selection and
- First value selection.

IV. ASYNCHRONOUS MULTIPROCESSOR SYSTEM DESCRIPTION

RAMPS can be described graphically in the following diagram where boxes represent CPU's and circles represent buffers.

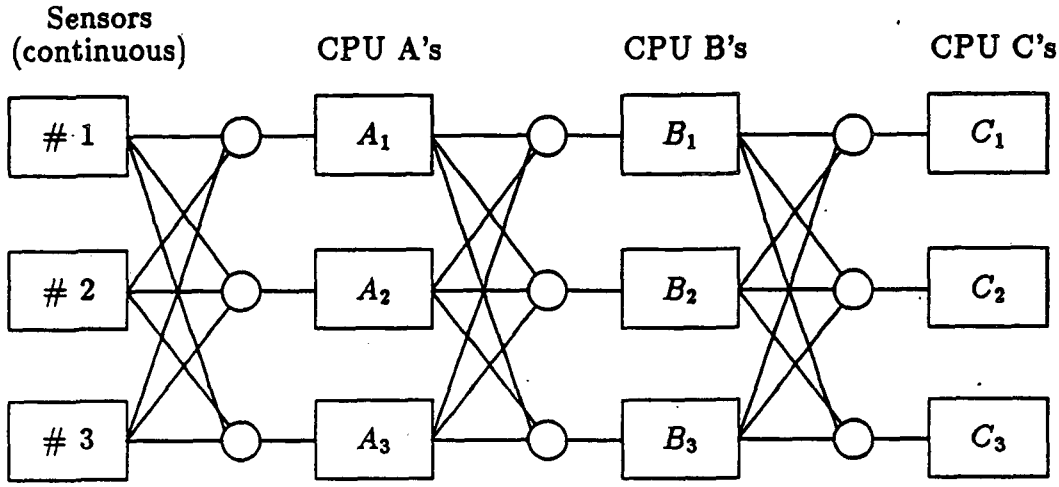


Fig. 4.1 Schematic Diagram of RAMP system

The major characteristics of this system are as follows:

- The first parallel sequence consists of sensors. These sensors give continuous outputs.

- Each CPU is operating independently of the others.

- Each CPU outputs at a fixed frequency to buffers which are read by the downstream CPU's.

- Parallel CPU's at the same sequential position perform the same task. Their execution times are slightly different (within a known tolerance and about a known nominal time.)

- Sequential CPU's in the same parallel pipeline perform different tasks.

There are three kind of faults which can occur in the RAMPS modules[1]:

- Faults that cause data alteration (Fig. 4.2)
- Faults that result in improper execution of the designed code (Fig. 4.3.)
- Faults that cause excessive timing differences in parallel CPU's.

The purpose of this study is to isolate and identify errors due to timing differences so that the other faults are not masked by the effect of the timing differences.

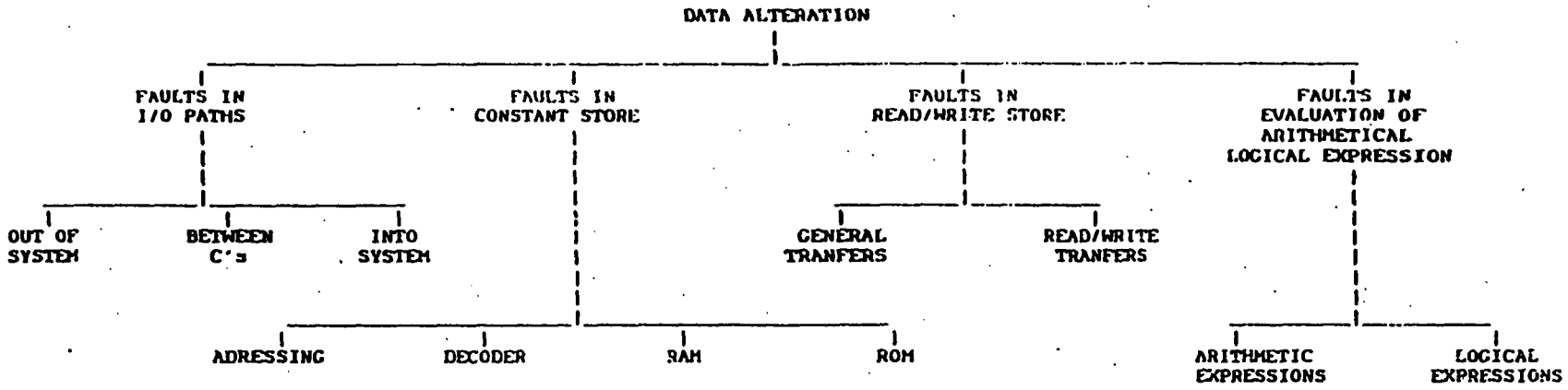


FIG. 4.2. FAULTS RESULTING IN DATA ALTERATION

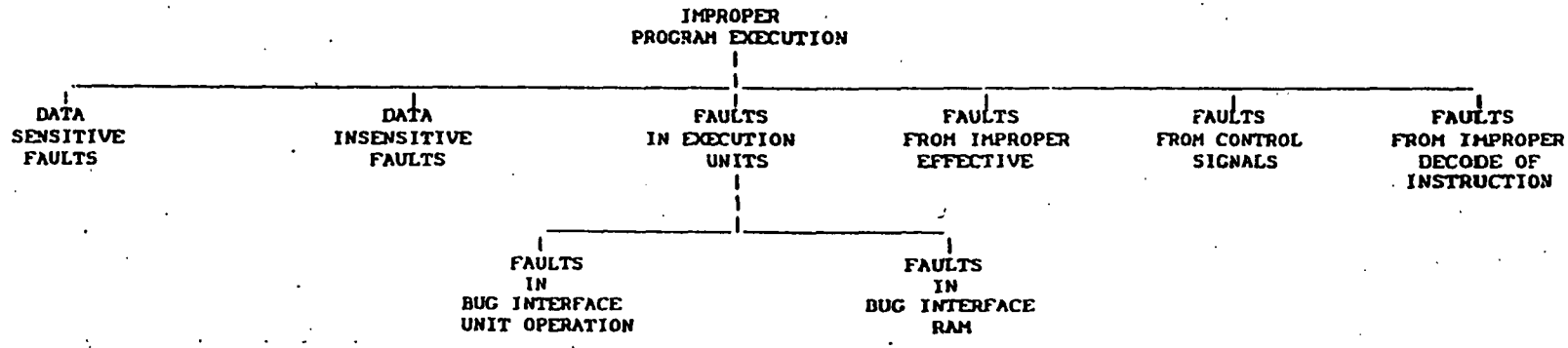


FIG. 4.3. FAULTS RESULTING IN IMPROPER PROGRAM EXECUTION

V. SIMULATION PROGRAM

A simulation of the RAMP system was implemented on a Micro-Vax II computer. The purpose of the simulation was to investigate the differences in the output of parallel CPU's due to differences in their internal clock rates. In order to illustrate the effect of these clock differences the following example is given.

- Each of two CPU's has a 5 Mega hertz (Mhz) clock that is different by 0.001%. The clock difference (ΔC) is then:

$$\Delta C = \frac{1}{5Mhz} 0.001\% = 2. \times 10^{-12} sec./clockcycle$$

- If the computation cycle (T) for the CPU's is 50 milliseconds (mS) the number of clock cycles that it will take to produce one computation cycle difference between the CPU's is:

$$ClockCycles = \frac{T}{\Delta C} = \frac{50 \times 10^{-3} sec./comp.cycle.}{2 \times 10^{-12} sec./clockcycle} = 25 \times 10^{9} clockcycles/comp.cycle$$

- The elapsed time (t) over which the CPU's will develope a one computation cycle (50 mS) asynchronisity is then:

$$t = \frac{clockcycles/comp.cycle}{clockcycle/sec.} = \frac{25 \times 10^{9}}{5 \times 10^{6}} = 5000.sec/comp.cycle = 83min.20sec.$$

Since most flights would run in excess of one hour, this problem will be significant. In order to facilitate the analysis and minimize the require computer time for the analysis, greatly exaggerated clock differences are used in the simulation.

In the simulation, each CPU solves an algorithm of the form:

$$X^{m}_{n+1} = \alpha X^{m}_{n} + \beta U_{n}$$

Where the supercript (m = A, B, C) denotes the parallel pipeline and the subcript (n = 1, 2, 3, ...) denotes the computation cycle number. The following table shows the coefficients of the algorithm used at each sequential CPU along with the computation cycle times for the three parallel CPU's at each sequential node.

	CPU-A			CPU-B			CPU-C		
	A1	A2	A3	B1	B2	B3	C1	C2	C3
time required	0.02	0.0202	0.0201	0.035	0.0352	0.0351	0.018	0.0182	0.0181
α	0.88			0.93			0.85		
β	0.12			0.07			0.15		

The structure and data flow of the simulation program is shown in Fig. 5.1. On each pass the Executive first calls the sensor subroutine to generate input data for the A CPU's. If any of the three parallel A CPU's are due to receive data on this pass, the CPU A subroutine is called and the results are stored in the output buffer of the appropriate A CPU. The same process is applied to the B and C CPU's on each pass.

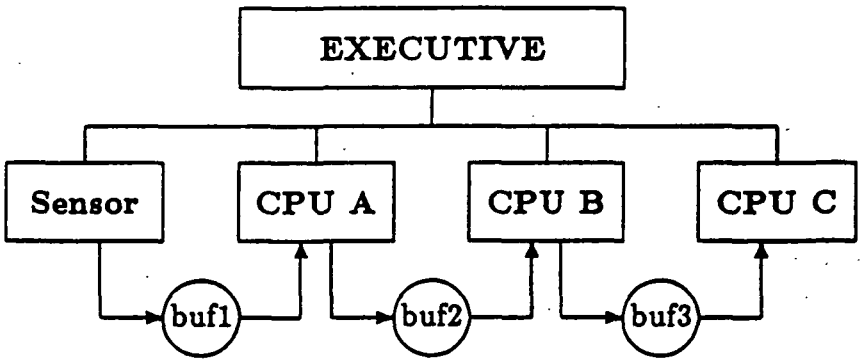


Fig. 5.1 Data flow diagram.

Figure 5.2 shows the internal logic of the Executive. Real time (t) is incremented by an amount DT chosen to be much smaller than the difference in computation cycle times. The next interrupt time for each of the j (j=1,2,3) CPU's at the A, B and C nodes is then computed. The sensor subroutine is called each real time pass to simulate a continuous sensor input to CPU-A. For purpose of the simulation, the sensor outputs are modeled as pure sinusoids. Random noise is added to each of the three sensor outputs. Current real time is then compared to the interrupt times for all CPU's and any CPU's requesting data on the current data pass are processed and the output stored in the appropriate buffer. Whenever a CPU is processed, an incremented computation cycle count $(i_A(j), i_B(j), i_C(j))$ is returned for use in computing its next interrupt time.

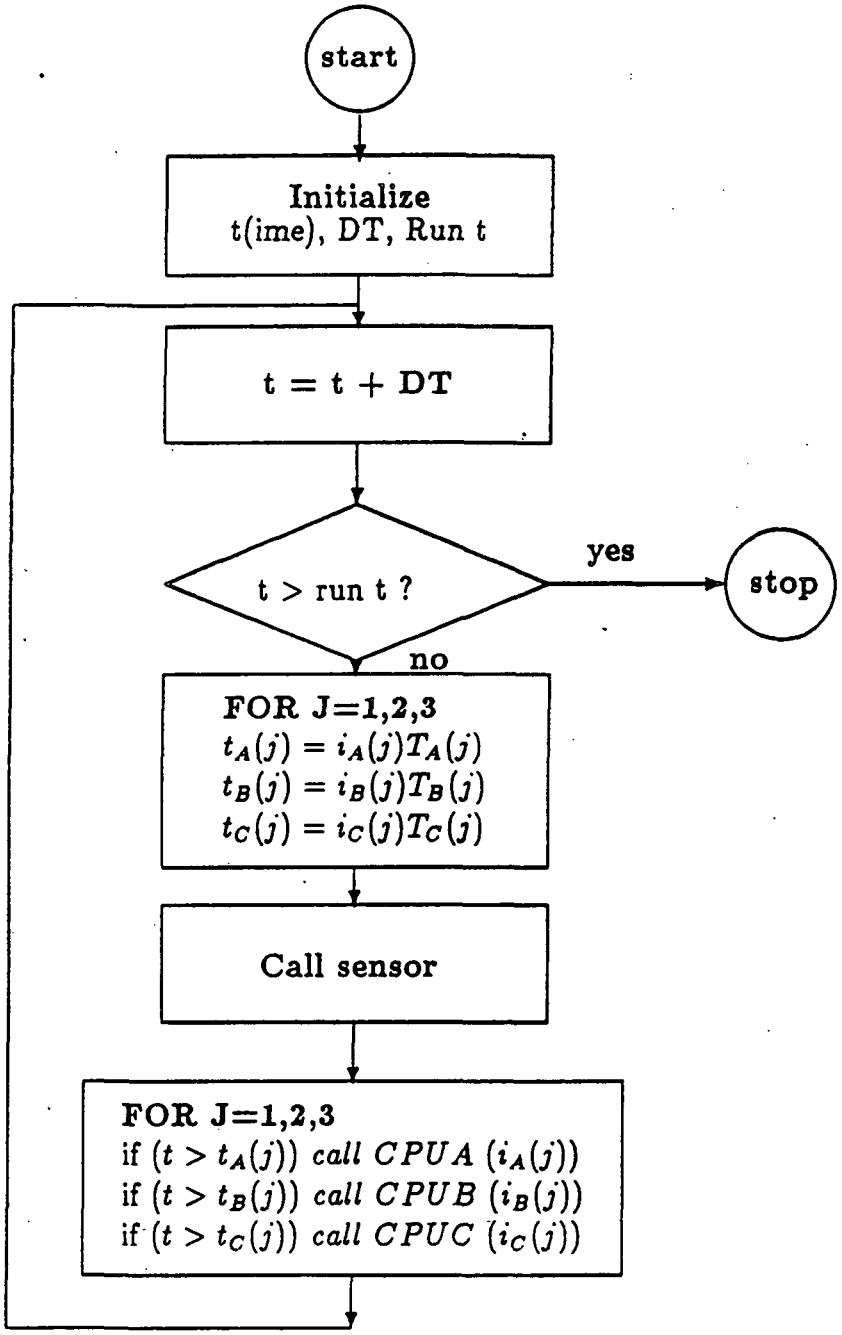


Figure 5.2: Executive Flowchart

VI. DEFINITION OF ERROR DUE TO ASYNCHRONISITY

Denoting the output of CPU-A1 as A1 and the output of CPU-A2 as A2 then the error may be expressed in two ways:

ERROR AT CPU-A1 AND CPU-A2 OUTPUT TIMES

The error, A1-A2, is computed at each output interval for either CPU. The error is seen to flip back and forth between two curves. One curve represents the error when CPU1 outputs and the other curve represents the error when CPU2 outputs. The result of this error definition is shown in Fig. 6.1.

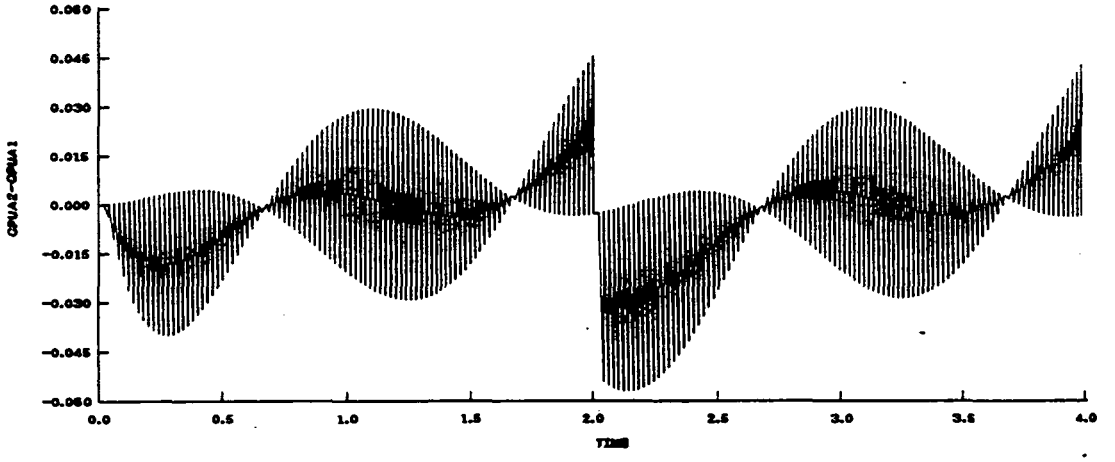


Fig. 6.1 Error at CPU-A1 and CPU-A2 Output Times

ERROR AT CPU-B INPUT TIMES

The error, A1-A2, is computed at points where CPU-B goes to the buffer to obtain the outputs of the A CPU's. Since the B CPU's have clock times that are out of synchronization with the A CPU's, B will sometimes sample when A1 has output most recently and sometimes when A2 has output most recently. The result is a sporadic oscillation between the two bounding error curves, as shown in Fig. 6.2.

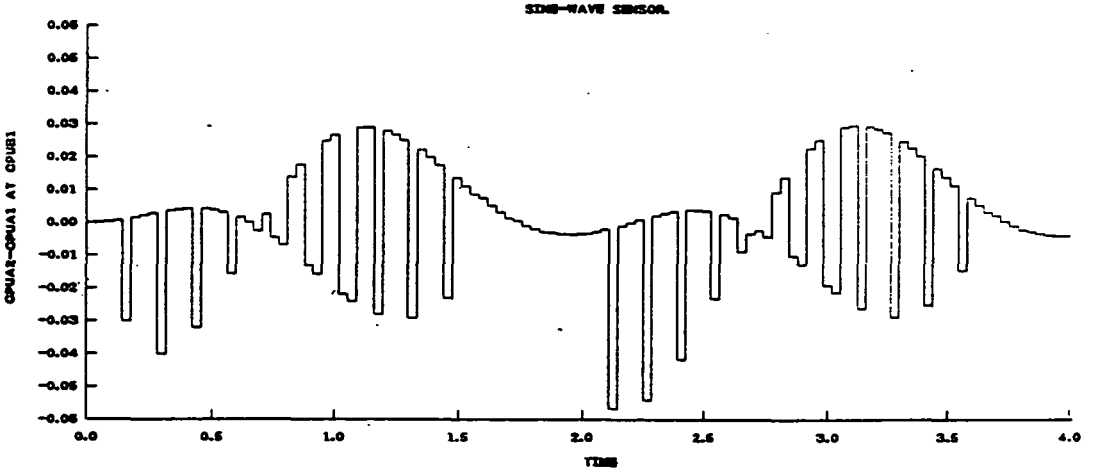
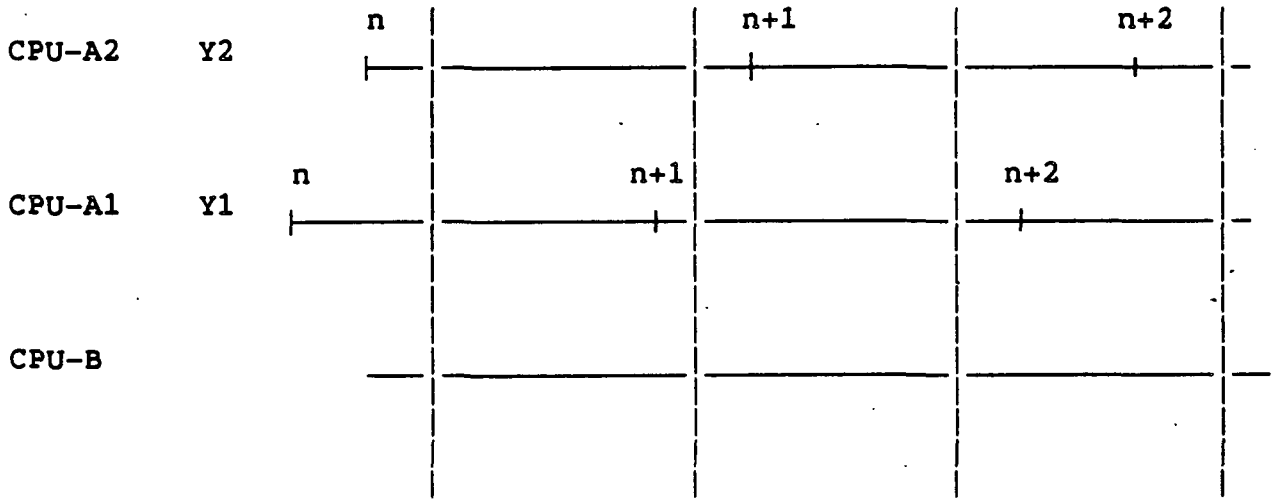


Fig. 6.2 Error at CPU-B Input times

The non-analytic nature of the error as seen by CPU-B significantly complicates the modeling process. This problem can be eliminated by defining the error such that the most recent output used in the error definition always comes from the same CPU. This assures that the error will always stay on the same bounding curve. This synchronization process is shown in Fig. 6.3. In this figure, the output times associated with CPU's A1, A2 and B are superimposed. The error associated with using the two most recent outputs, as seen by CPU B, is shown under the chart. If each CPU tags its output with a computation cycle number so that the error can be defined as the difference at the same computation cycles then the resulting synchronized error is as given in the second line under Fig. 6.3. Note that there is one input cycle for CPU B at which there is no new information available using this definition. The resulting error is shown in Fig. 6.4 and, as predicted, stays on the same bounding curve, providing a well defined and analytical function.



Continuous error : $Y_2(n)-Y_1(n)$ $Y_2(n)-Y_1(n+1)$ $Y_2(n+1)-Y_1(n+1)$ $Y_2(n+2)-Y_1(n+2)$

Synchronized error: $Y_2(n)-Y_1(n)$ - $Y_2(n+1)-Y_1(n+1)$ $Y_2(n+2)-Y_1(n+2)$

Fig. 6.3 Synchronized Error Definition

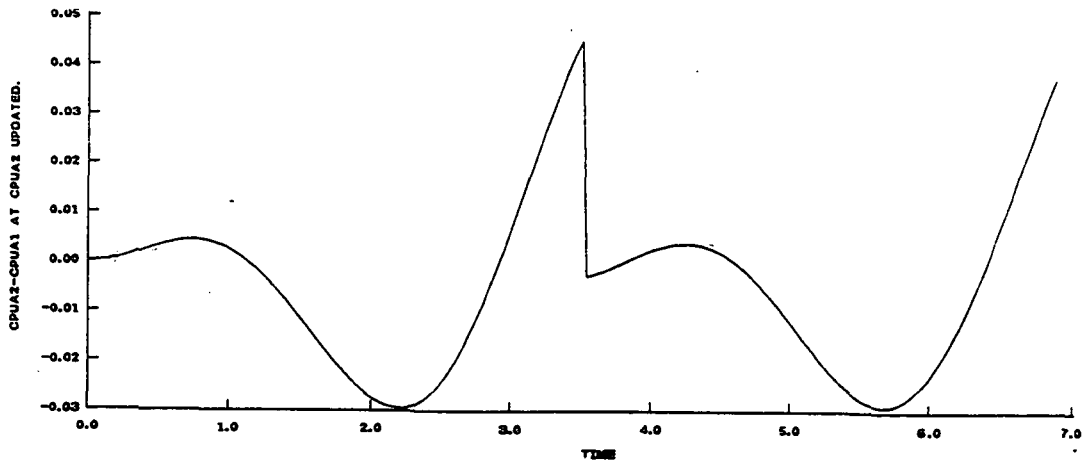


Fig. 6.4 Synchronized Error at Node A

VII. DERIVATION OF MATHEMATICAL MODEL OF ERROR

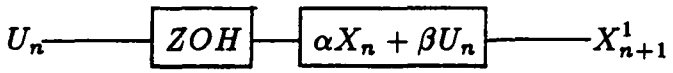
Assume that each CPU performs a first order linear differential equation of the form $\dot{X} = -aX + bU$. Hence the output in the discrete domain at cycle n+1 can be written as:

$$X_{n+1} = \alpha X_n + \beta U_n$$

where

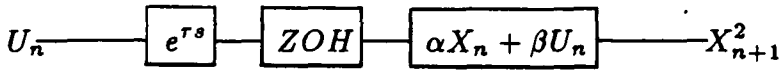
$\alpha = e^{-aT}, \beta = \frac{b}{a}(1 - e^{-aT})$
and T is the nominal sample interval.

Let CPU-1 be the standard CPU, the block diagram for CPU-1 is:



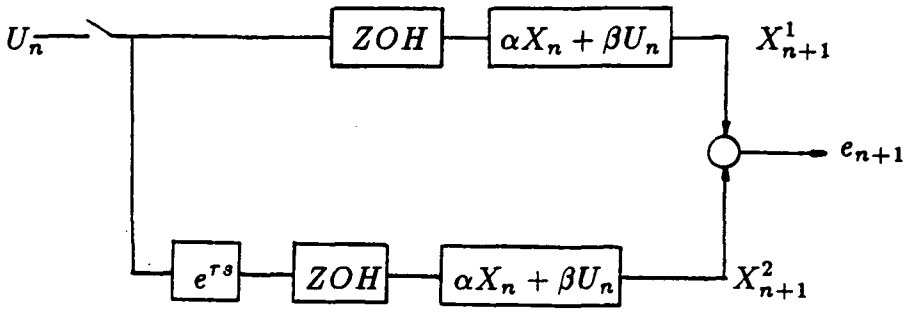
where the Zero Order Hold (ZOH) holds the input constant over the sample interval.

The difference in clock times between CPU-A1 and CPU-A2 can be considered as the time delay of CPU-A2 with respect to CPU-A1. The block diagram of CPU-A2 then can be drawn as follows:



where a pure time delay of magnitude τ has been added.

The model error now can be drawn by combining the above diagrams.



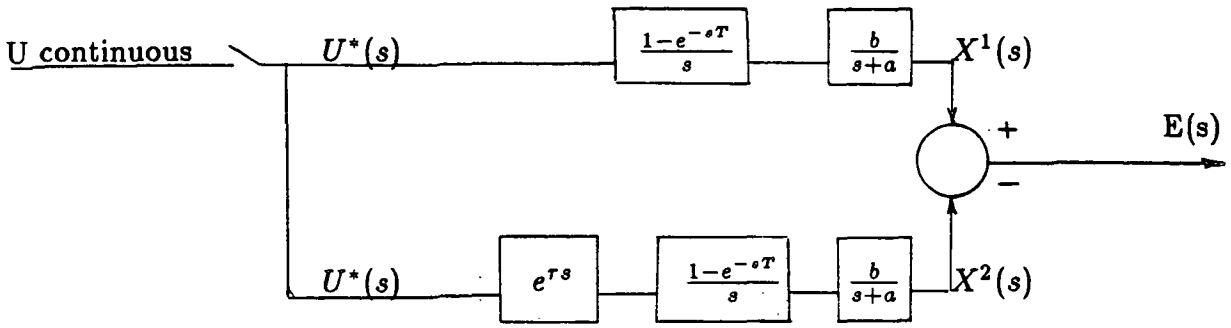
where

$$e_{n+1} = X^2_{n+1} - X^1_{n+1}$$
$$\tau = [T_2 - T_1]n$$

T1 : computational time of CPU-1
T2 : computational time of CPU-2
n : computational cycle number

The mathematical error model now can be derived as follows:

Using Laplace Transformation of the elements in the schematic gives:



The transfer function for CPU-1 is:

$$\frac{X^1(s)}{U^*(s)} = \frac{1-e^{-sT}}{s}\frac{b}{s+a} \tag{7.1}$$

and the transfer function for CPU-2 is:

$$\frac{X^2(s)}{U^*(s)} = \frac{1-e^{-sT}}{s}\frac{b}{s+a}e^{\tau s} \tag{7.2}$$

The transfer function for the error is then:

$$\frac{E(s)}{U^*(s)} = \frac{X^2(s)-X^1(s)}{U^*(s)} = \frac{1-e^{-sT}}{s}\frac{b}{s+a}[e^{\tau s}-1] \tag{7.3}$$

Where E(s) is the output resulting from the impulse train input $U^*(s)$. The impulse transform may be converted to a Z-transform by

1) Substituting $z = e^{sT}$ where possible.
2) Converting the remaining s terms to functions of e^{sT} by performing a complex convolution [3] integral of the transfer function with the Laplace transform of an impulse train $\left[\ \mathcal{L}[\delta(t)] = \frac{1}{1-e^{-Ts}}\right]$
3) Substituting $z = e^{sT}$ in the resulting equation.

Applying steps 1) and 2) give:

$$\frac{E(z)}{U(z)} = \frac{1}{2\pi j}\frac{z-1}{z}\oint_C \frac{1}{1-e^{-T(s-\lambda)}}\ \frac{b[e^{\tau\lambda}-1]}{\lambda(\lambda+a)}d\lambda \tag{7.4}$$

Using the residue theorem to solve this complex convolution integral gives:

$$\frac{E(z)}{U(z)} = \frac{z-1}{z} \sum_{res} \frac{1}{1-e^{-T(s-\lambda)}} \frac{b[e^{\tau\lambda}-1]}{\lambda(\lambda+a)} \tag{7.5}$$

In order to to facilitate evaluating the residues, a partial fraction expansion is applied to get:

$$\frac{E(z)}{U(z)} = \frac{z-1}{z} \sum_{res} \frac{b(e^{\tau\lambda}-1)}{1-e^{-T(s-\lambda)}} \frac{1}{a}\left(\frac{1}{\lambda} - \frac{1}{\lambda+a}\right) \tag{7.6}$$

Evaluating the residues, equation (7.6) becomes:

$$\frac{E(z)}{U(z)} = \frac{b}{a}\frac{z-1}{z}\frac{1-e^{-a\tau}}{1-e^{-T(s+a)}} \tag{7.7}$$

The final result is obtained by substituting $z = e^{sT}$ in (7.7) to get:

$$\frac{E(z)}{U(z)} = \frac{b}{a}\frac{z-1}{z-e^{-aT}}(1-e^{-a\tau}) \tag{7.8}$$

The difference equation for the error is then obtained from the Z-domain transfer function as:

$$E_{n+1} = e^{-aT}E_n + \frac{b}{a}(1-e^{-a\tau})(U_{n+1}-U_n) \tag{7.9}$$

Assuming the CPU's are initially synchronized, the phase shift, τ, beween the CPU's is a time varying function given by:

$$\tau = \frac{t}{T}\Delta T \tag{7.10}$$

where ΔT is the difference in the clock cycles between CPU-1 and CPU-2.

At τ = T the two CPU's have become one computation cycle out of phase. In order to avoid making the error model dependent on more than one previous error value, the error model is redefined at this point to be:

$$E^{+}_{n+1} = X^{2}_{n+1} - X^{1}_{n} \tag{7.11}$$

where the superscript "+" denotes the redefined error. For consistency with the redefined error, the phase shift, τ, is reset to zero:

$$\tau = 0 \tag{7.12}$$

Both the redefinition of the errors and the resetting of the phase shift to zero for consistancy with the redefined errors occur only when the phase shift equals one full computational cycle.

Writing the redefined error at time n (E_n^+) in terms of the previous error definition at time n (E_n^-) gives:

$$E_n^+ = X_n^2 - X_{n-1}^1 = X_n^2 - X_n^1 + X_n^1 - X_{n-1}^1 \tag{7.13}$$

$$E_n^+ = E_n^- + X_n^1 - X_{n-1}^1 \tag{7.14}$$

The error model is then propagated by equations (7.9) and (7.10) until τ= T. At this time the error is updated by the increment given in (7.14) and τ is reset to zero. Equations (7.9) and (7.10) then represent the error between CPU's 1 and 2 one computational cycle apart. Since this corresponds to the two most current values generated by the two CPU's, the error equation continually represents the difference in the two most current CPU outputs. The update process must be repeated whenever τ = T.

VIII. ANALYSIS OF ERROR MODEL

ERROR PREDICTION FOR A-NODE

The formulas derived in the previous section were implemented and compared with errors generated by the simulation program. Predicted error and measured error are plotted for comparison in Fig. 8.1.

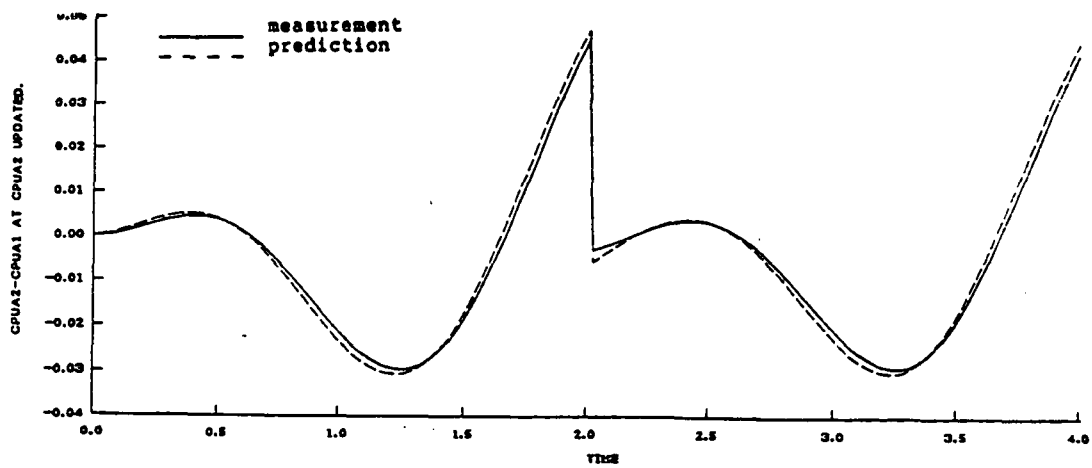


Fig. 8.1. Error Prediction at A Node

The predicted and measured error, based on the synchronized error definition, are seen to compare very well. The discontinuity at 2.0 seconds corresponds to the redefinition of the error when a full cycle of lag has built up and closely matches the measured value of the redefined error.

ERROR PREDICTION FOR B-NODE

In analyzing the error for the A-node CPU's, the effect of voting logic on the inputs to the A-nodes has been neglected because the relatively rapid sensor sampling rate results in the three sensor values being very similar. The results are therefore similar regardless of the input selected. This is not true for CPU's at the B and C nodes. There the sampling rates are considerably slower so the sampled outputs from the A CPU's will be significantly different as the effective phase lag, , increases. The two types of voting logic to be examined are select CPU-1 and Midvalue select.

Select CPU-1 Logic

The select CPU-1 logic chooses, as the default, the output of the CPU in the first pipeline at the previous node for input to the next node. The error in the B-node CPU's as seen at the C-node CPU's is shown in Fig. 8.2, where the input to the B-node CPU's has been based on the select CPU-1 logic. Fig. 8.3 shows the error, as seen at the C-node, using the synchronized error definition.

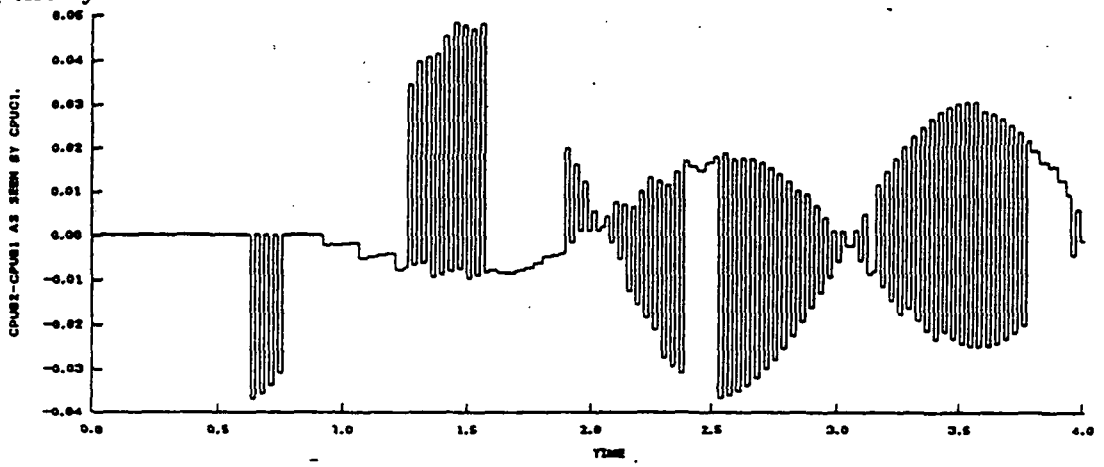


Fig. 8.2 Error at B-Node; Select CPU-1

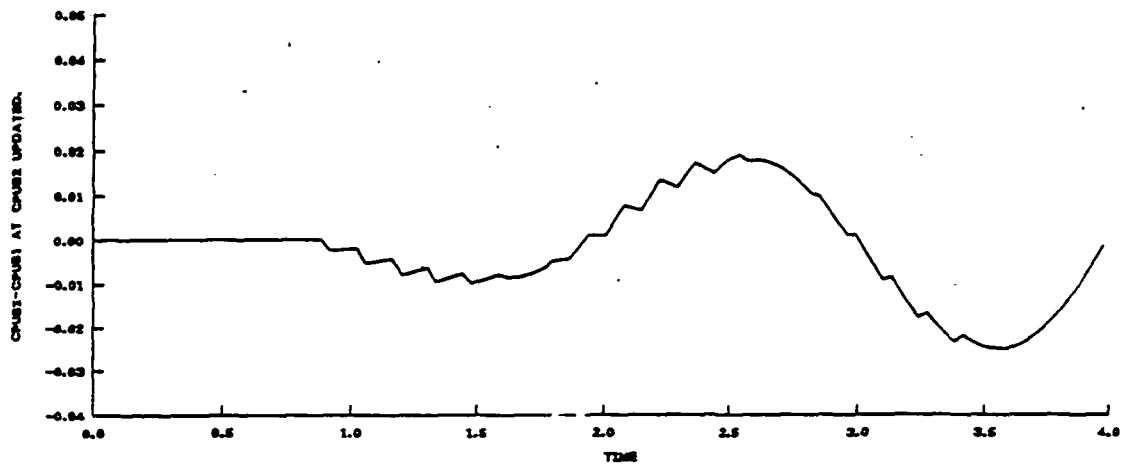


Fig. 8.3 Synchronized Error at B-Node; Select CPU-1

Although the error now is represented by only one of the two bounding curves, the function is highly discontinuous. This is due to the low sampling rate applied at the inputs to the B-node CPU's.

Midvalue Select Logic

The midvalue select logic chooses, as the default, the output of the CPU at the previous node whose value is in the middle of the output range of the CPU's at that node. The error in the B-node CPU's as seen at the C-node CPU's is shown in Fig. 8.4, where the input to the B-node CPU's has been based on the Midvalue select logic. Fig. 8.5 shows this error using the synchronized error definition.

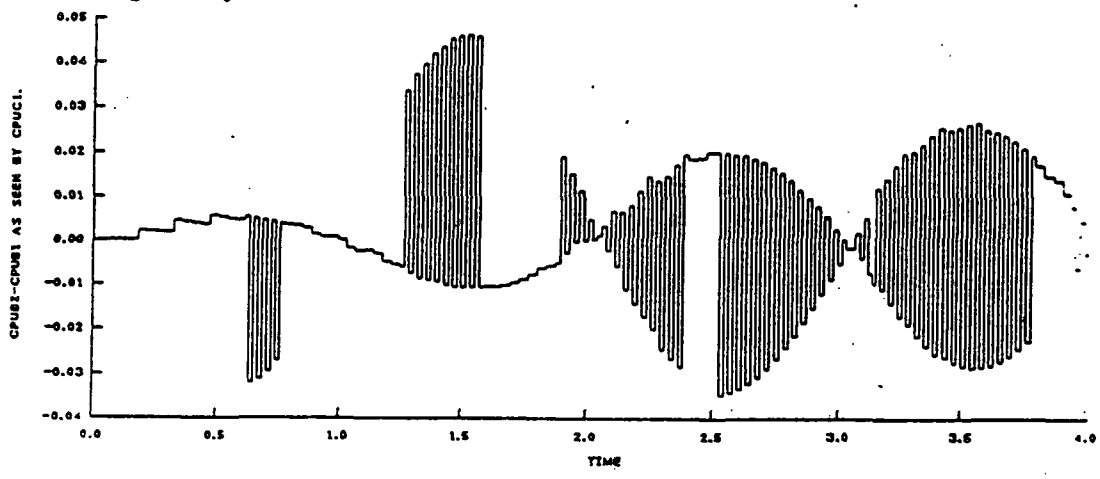


Fig. 8.4 Error at B-Node; Select Midvalue

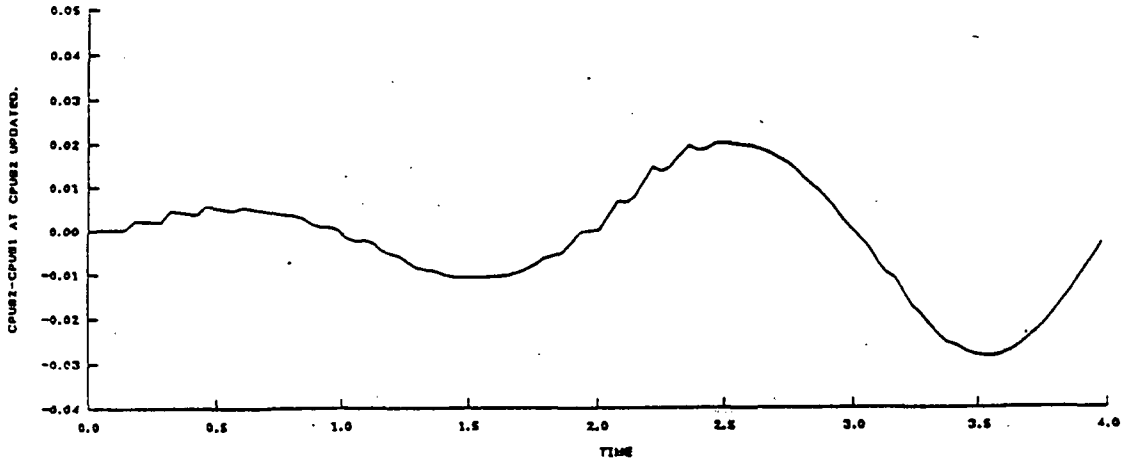


Fig. 8.5 Synchronized Error at B-Node; Select Midvalue

Comparing Fig. 8.5 to Fig. 8.3, it can be seen that the Midvalue select produces a smoother error curve than the select CPU-1 logic, so this will be used for the remaining analysis.

Input Smoothing

The error prediction for A-node CPU's is good because their inputs are from analog devices (sensors,) these inputs are smooth and continuous. The situation in B (and C) node CPU's is different since the inputs to B node CPU's, as obtained from the A-node CPU's,

are step functions. Since, the derived math model is based on smooth and continuous inputs the results will be degraded.

To resolve the problem of discontinuity, a first order hold was applied in each CPU to smooth its output. Figure 8.6 shows the measured and predicted error with a first order hold applied to the output of the B CPU's. The curve is smooth and, hence a good approximation is achieved with the error model.

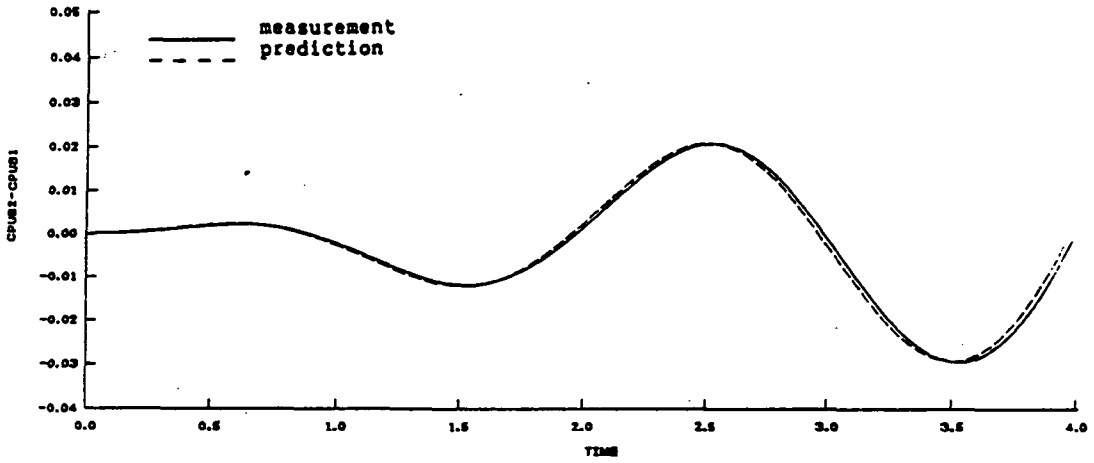


Fig. 8.6 Error Prediction at B-Node

ERROR PREDICTION FOR C-NODE

Figure 8.7 shows the error between the C-node CPU's. The strange shape is a result of two levels of sampling between the continuous input to the A-node CPU's and the output of the C-node CPU's. Figure 8.8 shows the synchronized error at the C-node and the error is seen to stay on the same bounding error curve. The effect of output smoothing at the A and B nodes is shown in Fig. 8.9 and compared with the model prediction. The results compare quite well.

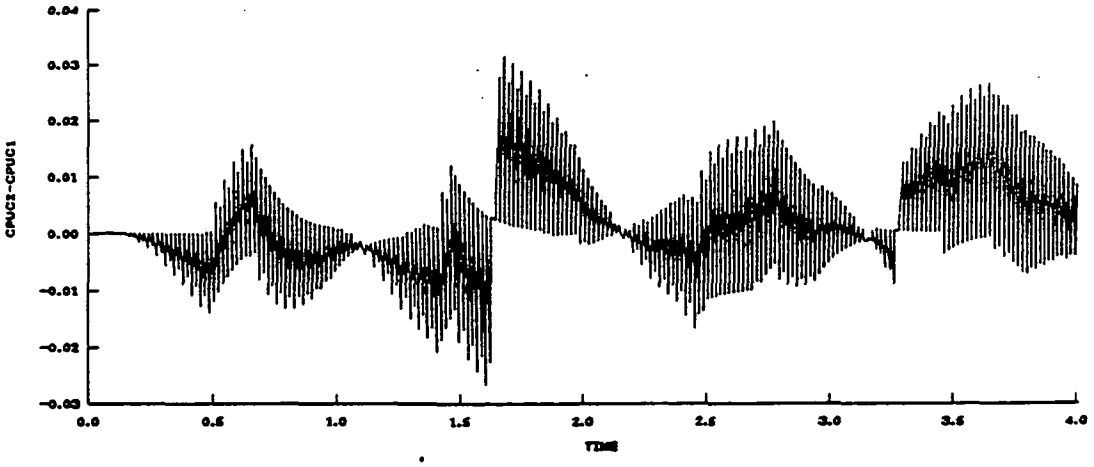


Fig. 8.7 Error at C-Node

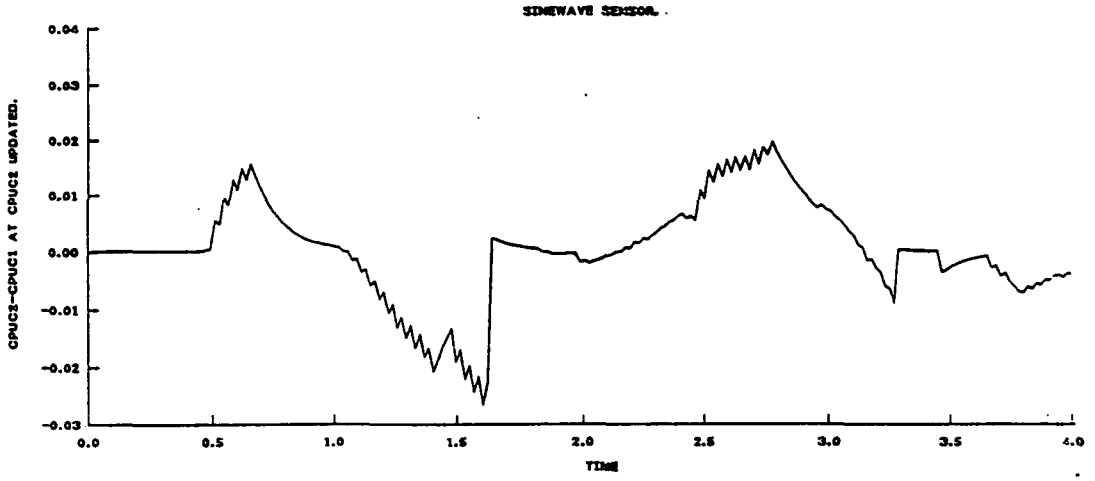


Fig. 8.8 Synchronized Error at C-node

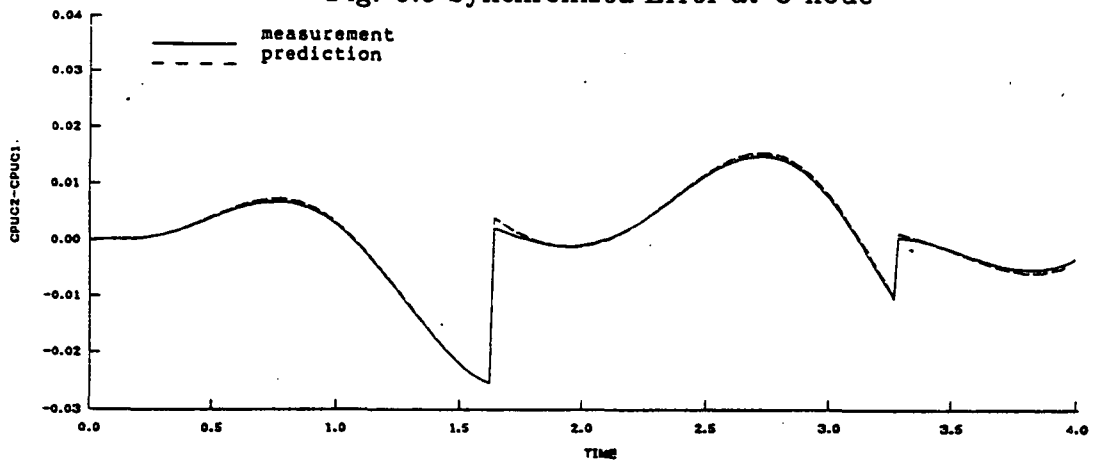


Fig. 8.9 Error Prediction at C-Node

In Fig. 8.9 the smoothing has been done by interpolation, which assumes that the next output is available at the time the smoothed output must be generated. Since this is generally not the case, a first order smoothing based on extrapolating the currently available data was implemented and the results are shown in Fig. 8.10 along with the predicted results. The comparison is still quite good and the implementation is now physically achievable.

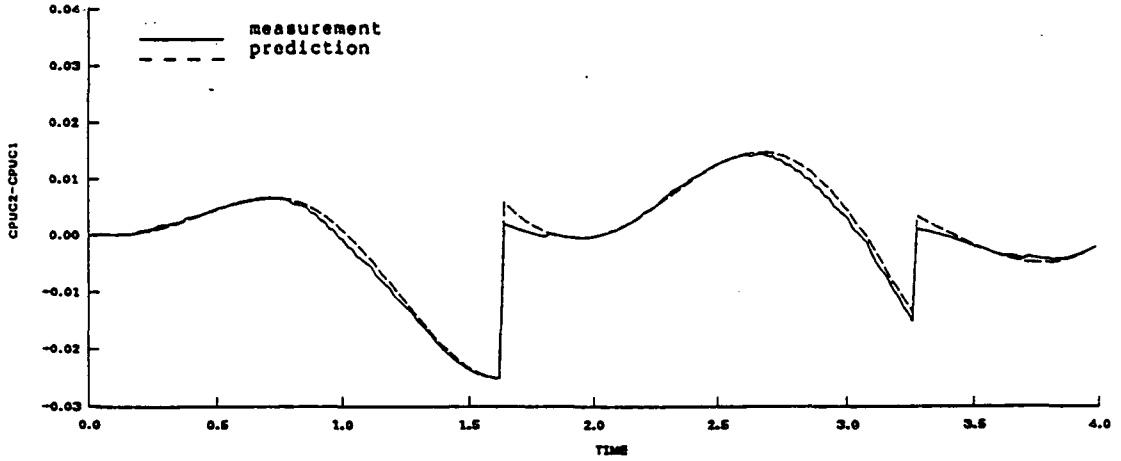


Fig. 8.10 Error Prediction at C-Node; extrapolation

SUMMARY OF CPU OPERATIONS

The following tasks must be performed each cycle by each CPU to allow estimation of the error due to asynchronisity given a knowledge of the difference in CPU clock rates (ΔT).

a. Estimate the error due to asynchronisity from upstream CPU's based on the inputs. b. Calculate the output. c. Smooth the output by using a first order hold.

These three tasks are shown in Fig. 8.11

Error Estimation
$E_{n+1} = e^{-aT}E_n + \frac{b}{a}(1 - e^{-a\tau})(U_{n+1} - U_n)$
Output Computation
$X_{n+1} = \alpha X_n + \beta U_n$
Output smoothing
$X(t) = X_n + \frac{X_{n+1} - X_n}{T} (t - nT)$

Fig. 8.11 CPU diagram.

IX. PARAMETER IDENTIFICATION

Using the error formula developed in section VII, the difference in CPU clock interval, ΔT, can be identified using the Output Error Method [4]. The identified ΔT then will be used to track the error due to asynchronisity. The mathematical derivation of the identification algorithm and the results are presented in this section.

MATHEMATICAL DERIVATION

In order to simplify the algorithm, the derivation assumes that the phase shift does not exceed one computation cycle. This is insured in practice by resetting the error equation whenever the phase reaches a full computation cycle.

Given a dynamic model of the output difference due to asynchronisity:

$$E_{n+1} = e^{-aT}E_n + \frac{b}{a}(1 - e^{-a\tau})(U_{n+1} - U_n) \tag{9.1}$$

$$\tau = (\frac{t}{T})\Delta T = n\Delta T \tag{9.2}$$

and given a measurement, E_n^m, of the difference at cycle n, a measurement error may be defined as:

$$e_n = E_n^m - E_n(\Delta T) \tag{9.3}$$

Where the estimated difference, $E_n(\Delta T)$, is a function of the difference in clock intervals. A cost function is now written in terms of the measurement error as:

$$J(\Delta T) = \frac{1}{2}\sum_{i=0}^{N} e_n^2(\Delta T) \tag{9.4}$$

Expanding J as a Taylor's series in ΔT about an apriori estimate, $\overline{\Delta T}$, gives:

$$J(\Delta T) \approx J(\overline{\Delta T}) + \frac{\partial J}{\partial \Delta T}[\Delta T - \overline{\Delta T}] + \frac{1}{2}\frac{\partial^2 J}{\partial(\Delta T)^2}[\Delta T - \overline{\Delta T}]^2 \tag{9.5}$$

Taking the differential of J due to a change in ΔT gives:

$$J(\Delta T) - J(\overline{\Delta T}) = \left[\frac{\partial J}{\partial \Delta T} + \frac{\partial^2 J}{\partial(\Delta T)^2}[\Delta T - \overline{\Delta T}]\right](\Delta T - \overline{\Delta T}) \tag{9.6}$$

The necessary condition for J to be minimum with respect to ΔT is $\frac{\partial J}{\partial(\Delta T)} = 0$. This gives:

$$\frac{\partial J(\Delta T)}{\partial \Delta T} = \frac{\partial J(\overline{\Delta T})}{\partial(\Delta T)} + \frac{\partial^2 J(\overline{\Delta T})}{\partial(\Delta T)^2}[\Delta T - \overline{\Delta T}] = 0 \tag{9.7}$$

or

$$\Delta T = \overline{\Delta T} - \left[\frac{\partial^2 J(\overline{\Delta T})}{\partial(\Delta T)^2}\right]^{-1}\frac{\partial J(\overline{\Delta T})}{\partial \Delta T} \tag{9.8}$$

Taking the derivative of equation (9.4) with respect to ΔT at $\overline{\Delta T}$ gives:

$$\frac{\partial J(\overline{\Delta T})}{\partial \Delta T} = \sum_{n=0}^{N} e_n(\overline{\Delta T})\frac{\partial e_n(\overline{\Delta T})}{\partial \Delta T} \tag{9.9}$$

$$\frac{\partial^2 J(\overline{\Delta T})}{\partial(\Delta T)^2} \approx \sum_{n=0}^{N}\left[\frac{\partial e_n(\overline{\Delta T})}{\partial \Delta T}\right]^2 \tag{9.10}$$

and from 9.3 and 9.1-9.2 we get:

$$\frac{\partial e_n(\overline{\Delta T})}{\partial(\Delta T)} = -\frac{\partial E_n(\overline{\Delta T})}{\partial(\Delta T)} \tag{9.11}$$

and

$$\frac{\partial E_{n+1}(\overline{\Delta T})}{\partial(\Delta T)} = e^{-aT}\frac{\partial E_n(\overline{\Delta T})}{\partial \Delta T} + nbe^{-na\overline{\Delta T}}[U_{n+1} - U_n] \tag{9.12}$$

The complete procedure, based on updating the estimate of ΔT every N samples, is summarized below.

1) Propagate error and error sensitivity up to cycle n = N.

$$e_n = E_n^m - E_n$$

Resynchronization of the computation cycles of the two CPU's is accomplished by resetting the index, n, to zero whenever the phase shift ($n\Delta T$) equals one computation cycle (T). At this time, the error is also redefined as described in Equation 7.14. The equations for propagating the error and error sensitivity are then:

$$E_{n+1} = e^{-aT}E_n + \frac{b}{a}(1 - e^{-an\overline{\Delta T}})(U_{n+1} - U_n); \qquad E_0 = 0.$$

$$\frac{\partial e_n}{\partial \Delta T} = -\frac{\partial E_n}{\partial \Delta T}$$

$$\frac{\partial E_{n+1}}{\partial(\Delta T)} = e^{-aT}\frac{\partial E_n}{\partial \Delta T} + nbe^{-na\overline{\Delta T}}[U_{n+1} - U_n]; \qquad \frac{\partial E_0}{\partial \Delta T} = 0.$$

2) Generate first and second gradient of cost function for N cycles

$$\frac{\partial J}{\partial \Delta T} = \sum_{n=0}^{N} e_n \frac{\partial e_n}{\partial \Delta T}$$

$$\frac{\partial^2 J}{\partial(\Delta T)^2} = \sum_{n=0}^{N}\left(\frac{\partial e_n}{\partial \Delta T}\right)^2$$

3) Generate estimate of best ΔT over N cycles.

$$\Delta T = \overline{\Delta T} - \left(\frac{\partial^2 J}{\partial(\Delta T)^2}\right)^{-1}\frac{\partial J}{\partial \Delta T}$$

SUMMARY OF CPU OPERATION

The total operation required for tracking the difference in CPU outputs due to differences in the clock intervals is shown below

For previous CPU's (j = 1,2,3)

Estimate Error due to Asynchronisity/Identify Clock Difference

$$\Delta T = \overline{\Delta T} - \left(\frac{\partial^2 J}{\partial J^2}\right)^{-1} \frac{\partial J}{\partial \Delta T}$$

where

$$\frac{\partial J}{\partial \Delta T} = \sum_{n=0}^{N} e_n \frac{\partial e_n}{\partial \Delta T}; \qquad \frac{\partial^2 J}{\partial (\Delta T)^2} = \sum_{n=0}^{N} \left(\frac{\partial e_n}{\partial \Delta T}\right)^2$$

and

$$e_n = E_n^m - E_n(\Delta T); \qquad \frac{\partial e_n}{\partial \Delta T} = -\frac{\partial E_n}{\partial \Delta T}$$

where

$$E_{n+1} = e^{-aT} E_n + \frac{b}{a}(1 - e^{-a\tau})(U_{n+1} - U_n)$$

$$\tau = n\Delta T; \qquad 0 < \tau < T$$

and

$$\frac{\partial E_{n+1}}{\partial (\Delta T)} = e^{-aT} \frac{\partial E_n}{\partial \Delta T} + nbe^{-na\overline{\Delta T}}[U_{n+1} - U_n]$$

Select Input (j = 1,2,3)

If $e^j < \epsilon$ U = Midvalue of previous CPU output

Else Fail CPU^j

Compute Output

$$X_{n+1} = \alpha X_n + \beta U_n$$

Smooth output

$$X(t) = X_n + \frac{X_n + 1 - X_n}{T}(t - nT)$$

X. RESULTS OF PARAMETER IDENTIFICATION USING SIMULATION

The identification algorithm was tested by running the simulation with a specified ΔT between the clocks. The identification algorithm was then applied to the output data, initially assuming a ΔT that was different from the one used in the simulation. The algorithm would be considered successful if the estimated ΔT converged to the actual value and the corresponding estimate at the CPU output difference tracked the measured value accurately.

In order to examine the stability of the algorithm, the test was first run with an initial estimate of ΔT that was equal to the actual value. The dynamic model was not updated with the estimated ΔT. Figure 10.1 shows the history of the ΔT estimate. It is seen to initially depart from the correct value but to converge back to the correct value after about 35 samples and to hold that value from then on. The corresponding comparison of the CPU difference and its estimate is shown in Fig. 10.2 and is seen to be excellent, as expected since the correct value of ΔT is always used.

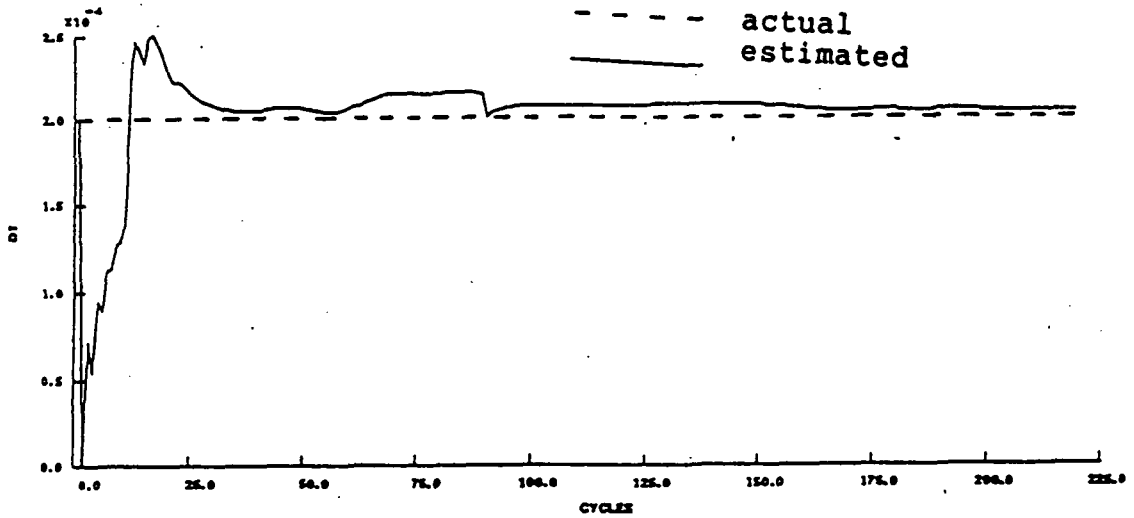


Fig. 10.1 ΔT Estimate ($\Delta T(0) = \Delta T^*$; No Feedback)

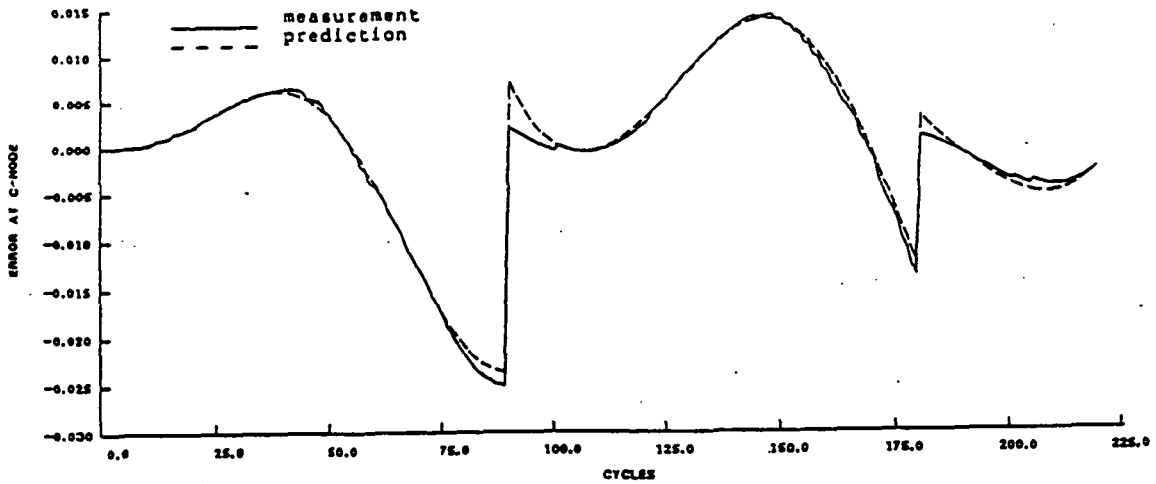


Fig. 10.2 Error Estimate ($\Delta T(0) = \Delta T^*$; No Feedback)

The second test was to initiate the identification with an estimate of ΔT that was half the actual value. Again the identified value was not used to update the dynamic model. The result of the identification is shown in Fig 10.3. Here we see that the correct value has again been achieved within 35 samples. Now, however, the estimate degrades significantly at the discontinuity associated with the error redefinition. This is due to the fact that the estimate is not being fed back to the model to allow the model output to converge to the measured CPU difference as can be seen from a comparison of the estimated and actual CPU differences in Fig 10.4.

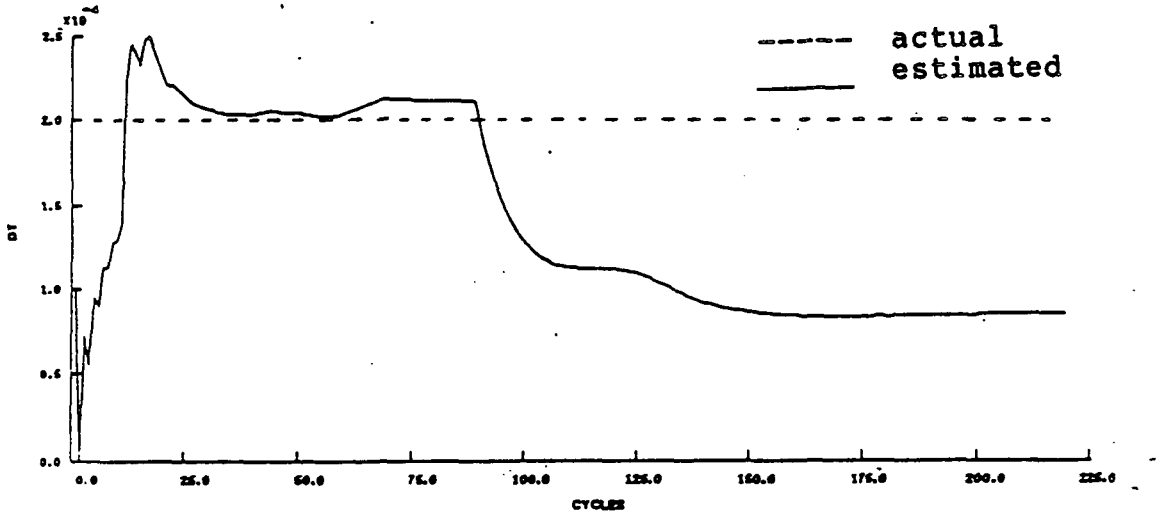


Fig. 10.3 ΔT Estimate ($\Delta T(0) = \frac{1}{2}\Delta T^*$; No Feedback)

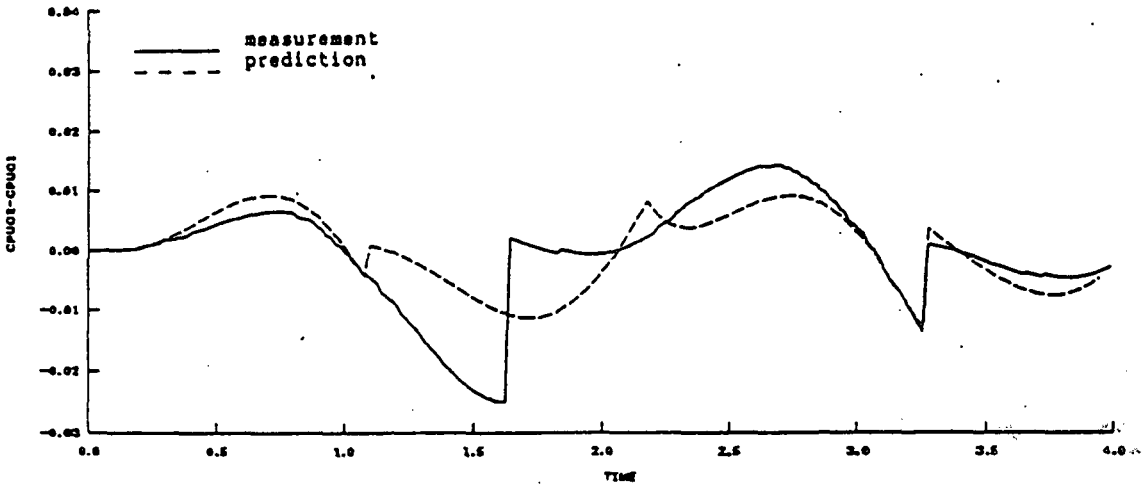


Fig. 10.4 Error Estimate ($\Delta T(0) = \frac{1}{2}\Delta T^*$; No Feedback)

The third test added the updating of the model with the identified ΔT while using the correct initial ΔT for the parameter estimator. The update was performed every 35 cycles since this number was shown to be adequate to achieve a steady state estimate. The results of the identification are shown in Fig 10.5 and the corresponding comparing of actual and estimated CPU differences is shown in Fig. 10.6. Now the identification is quite good until the discontinuity is reached. After this point, the accuracy degrades. This is apparently because the discontinuity generates large transients in the identification and the 35 sample update interval falls in the middle of this transient.

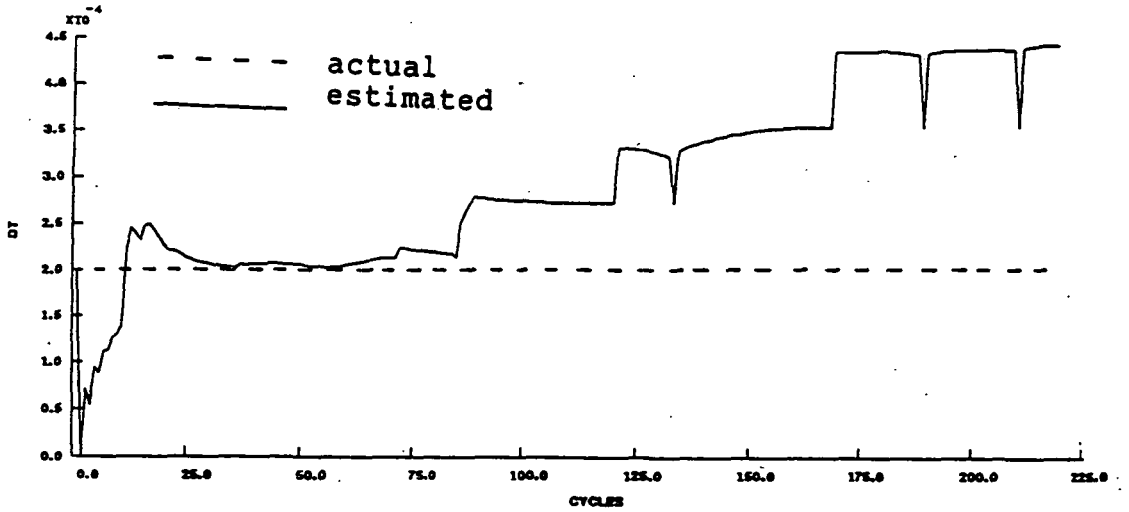


Fig. 10.5 ΔT Estimate ($\Delta T(0) = \Delta T^*$; Full Feedback, Update Every 35 Cycles)

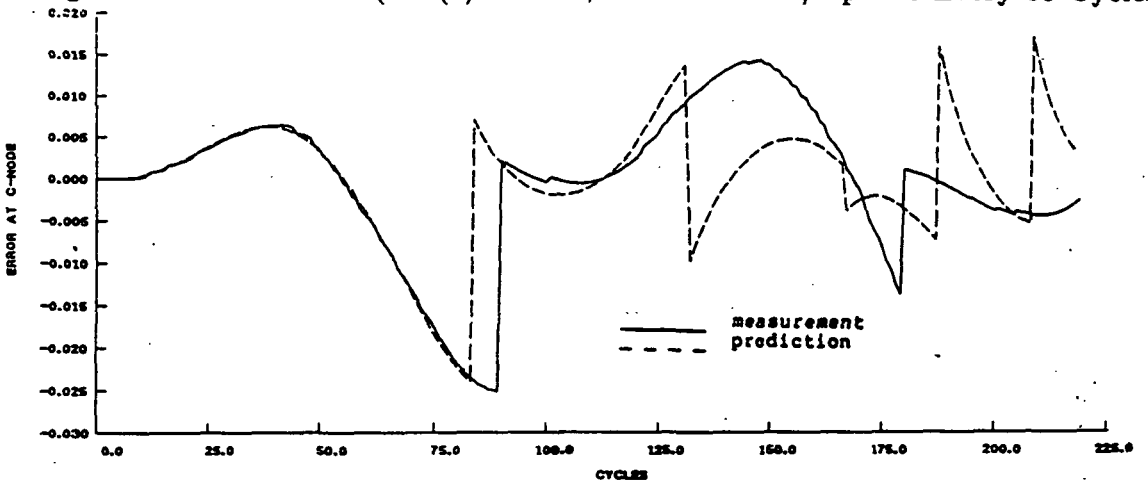


Fig. 10.6 Error Estimate ($\Delta T(0) = \Delta T^*$; Full Feedback, Update Every 35 Cycles)

In order to avoid this problem, a new criteria for the update was generated. The first update occurs automatically after 35 cycles. Successive recursive estimates of ΔT are compared and when the change in these estimates changes sign, it is assumed that the result is close to steady state and the model is updated with the currently identified value of ΔT. As a further prcaution, the update is only made if the magnitude of the change is below the magnitude of the previous update. The results of this approach are shown in Fig. 10.7 and 10.8. The identified ΔT and the output of the model both match the simulation values very well. A further improvement was achieved by reducing the magnitude of the update by a scale factor to allow a smoother convergence. The update algorithm is then:

$$\Delta T = \Delta T - \overline{\Delta T} = \frac{1}{1+(r)^k} \left[\frac{\partial^2 J(\overline{\Delta T})}{\partial(\Delta T)^2}\right]^{-1} \frac{\partial J(\overline{\Delta T})}{\partial \Delta T} \qquad (10.1)$$

where r is a value less than 1.0 and k is the update number. This scale factor allows the amount of the update to be gradually increased from 50% to 100% of the actual estimate over a period of time and improves the stability of the algorithm. The output of the model corresponding to this modified update algorithm is shown in Fig. 10.9 and is seen to match the measured CPU difference quite well.

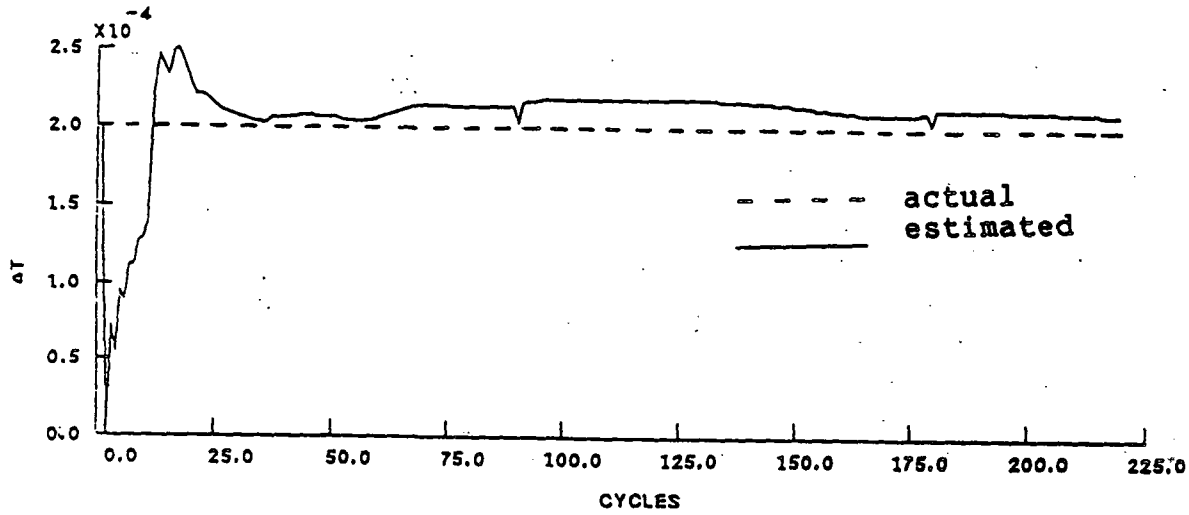


Fig. 10.7 ΔT Estimate ($\Delta T(0) = \Delta T^*$; Full Feedback, Update at Steady State)

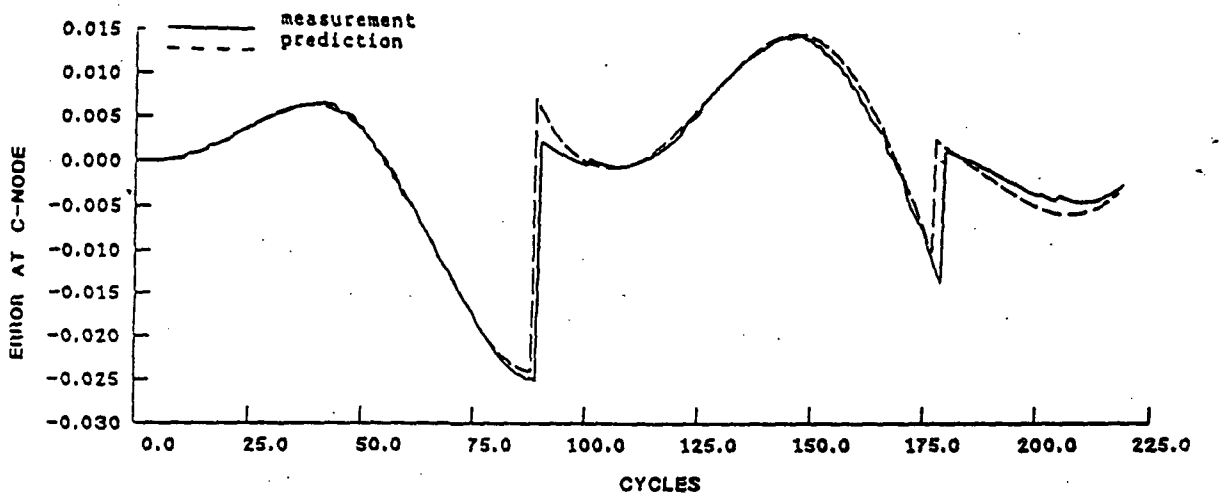


Fig. 10.8 Error Estimate ($\Delta T(0) = \Delta T^*$; Full Feedback, Update at Steady State)

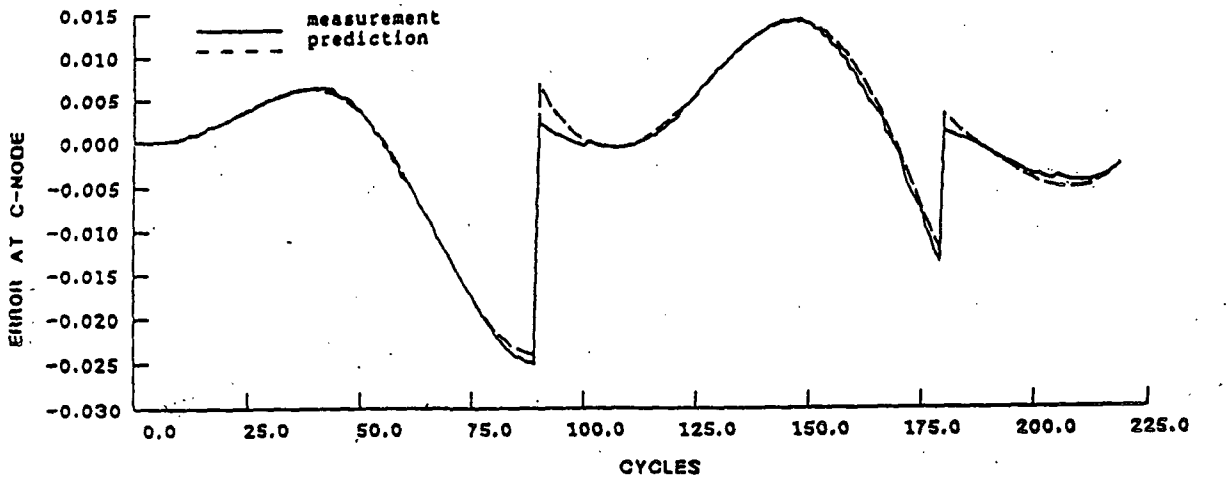


Fig. 10.9 Error Estimate ($\Delta T(0) = \Delta T^*$; Scaled Feedback, Update at Steady State)

As a final test of the algorithm, the previous procedure was run using an initial estimate of ΔT that was 50 % greater than the actual value. The update logic and update scale factor were applied as in the previous case and the results are shown in Figures 10.10 and 10.11. The identification algorithm and model output both track the simulation value quite well.

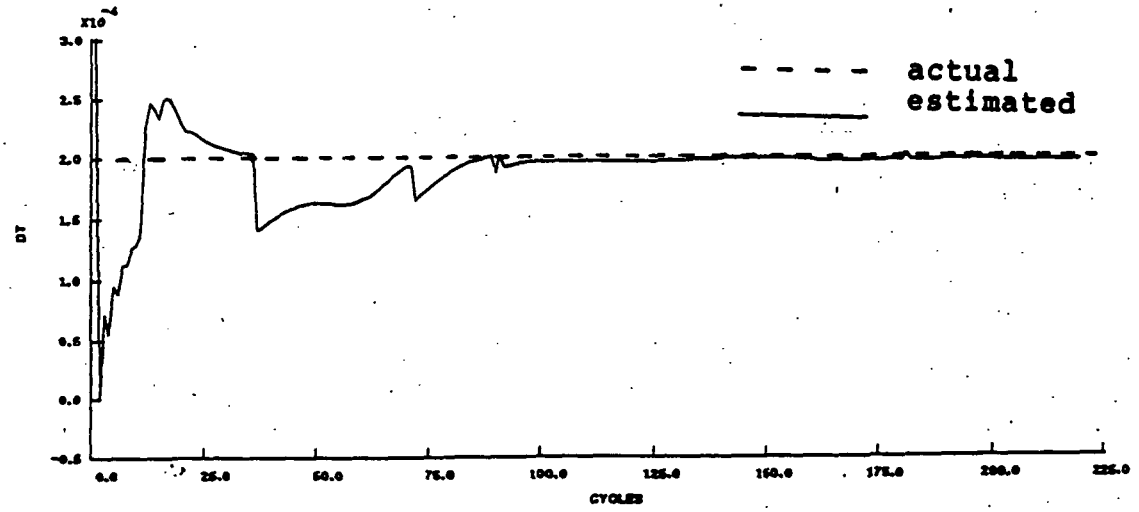


Fig. 10.10 ΔT Estimate ($\Delta T(0) = 1.5\Delta T^*$; Scaled Feedback, Update at Steady State)

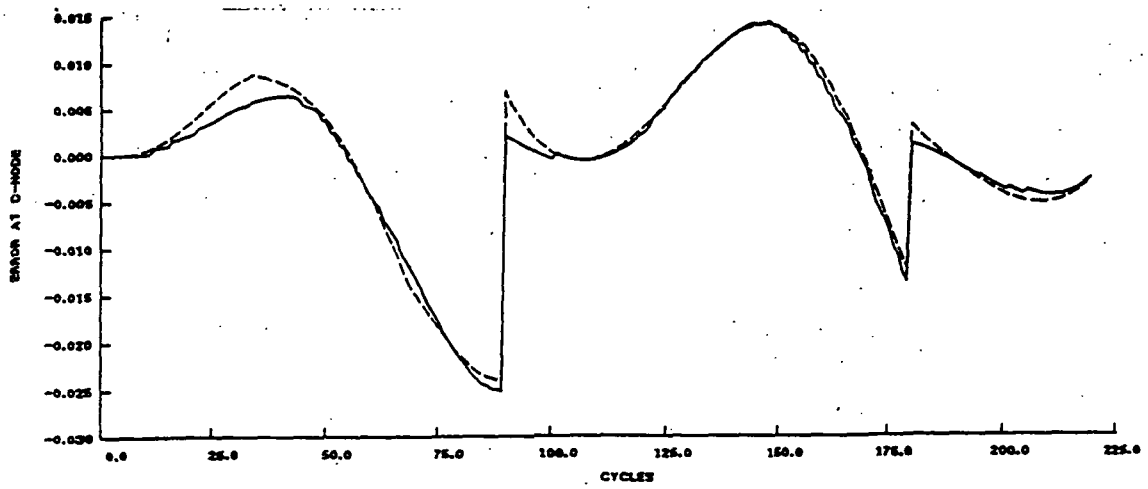


Fig. 10.11 Error Estimate ($\Delta T(0) = 1.5\Delta T^*$; Scaled Feedback, Update at Steady State)

XI. SUMMARY OF THE RESULTS OF THIS STUDY

The following accomplishments were achieved in this study

a. Definition of Analytic Error Function. The error between two CPU's was defined so as to produce an analytically describable function.

b. Error Model Derivation. An analytical model was developed to estimate the error between CPU's as a function of the difference in CPU clocks. The error model is based on a smooth, continuous input to each CPU; therefore each CPU must perform a first order hold to smooth its output.

c. Accuracy of Error Model Demonstration. The estimated error was shown to be within 2% of the measured error when the difference in clock intervals is known.

d. Identification of Difference in CPU Clock Interval . Since the error formula depends only on the difference in CPU clock interval, ΔT, true ΔT can be regressively tracked by applying Parameter Identification methods. Identified ΔT was shown to be within 5% error of the true ΔT after 100 cycles.

e. Updating of Model with identified Clock Intervals . Identified ΔT was fed back into the model and was successfully used to track the error due to asynchronisity.

XII. CONCLUSIONS

The ability to model and track the effect of different clock intervals on the relative output of redundant CPU's has been demonstrated using a computer simulation of the Redundant Asynchronous Multiprocessor System (RAMPS). This capability effectively eliminat the problem of asynchronisity on identifying failures without compromising the autonomy of each CPU. The high level of reliability associated with a RAMP system is therefore maintained. The identified clock differences could also be used to modify the computation algorithms in each CPU to keep the outputs synchronized, again without sacrificing their independence.

XIII. RECOMENDATIONS FOR FURTHER RESEARCH

The identification algorithm used in this study was based on the Output Error method. The use of Maximum Likelihood identification may allow for more rapid and robust tracking of the clock differences with only a slight increase in computational requirements. The use of this technique should therefore be investigated.

The algorithm modeled in each CPU for this study was a single input/single output algorithm. This algorithm was also asymptotically stable. In order to assure generality in

the application of this methodology, the use of multi-input multi-output algorithms that are not asymptotically stable should also be investigated.

The use of the identified clock difference to modify the computation algorithm to synchronize the CPU's could keep the error within small bounds without sacrificing CPU autonomy. This technique should be investigated.

The ultimate test of the approach will be to implement it in the RAMPS hardware rather than simulate a multiprocessor system on a single CPU.

REFERENCES

1. "Investigation of Redundant Asynchronous Microprocessors for High Authority Auto Flight Control", University of Southern Colorado, January, 1984.

2. "Distributed Asynchronous Microprocessor Architectures in Fault Tolerant Integrated Flight System", AIAA Computers in Aerospace Conference, Hartford, Connecticut, October, 1983.

3 Roberto Saucedo, Earl E. Schiring, "Introduction to Contnuous and Digital Control System", The Macmillan Company, New York, 1968.

4. Mehra, R. E., Stepner, D. E. and Tyler, J. S., " A Generalized Method for the Identification of Aircraft Stability and Control Derivatives from Flight Test Data", Proceedings of the Thirteenth Joint Automatic Controls Conference, Stanford, California, Aug., 1972.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-177427		

4. Title and Subtitle	5. Report Date
Dynamic Modelling and Estimation of the Error Due to Asynchronism in a Redundant Asynchronous Multiprocessor System	May, 1986
	6. Performing Organization Code: FSF

7. Author(s)	8. Performing Organization Report No.
Loc c. Huynh and R. W. Du Val	
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Advanced Rotorcraft Technology, Inc. 1804 Stierlin Road, Suite 210 Mountina View, CA 94043	A30146C(HMK)

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics & Space Administration Ames Research Center Moffett Field, CA 94035	Contractor Report
	14. Sponsoring Agency Code: RTOP #505-66

15. Supplementary Notes

Point of Contact: Technical Monitor, James Howard, MS 210-5
NASA Ames Research Center
Moffett Field, CA 94035 (415) 694-5941

16. Abstract

The use of Redundant Asynchronous Multiprocessor System to achieve Ultra reliable Fault Tolerant Control Systems shows great promise. The development has been hampered by the inability to determine whether differences in the outputs of redundant CPU's are due to failures or to accrued error built up by slight differences in CPU clock intervals. This study derives an analytical dynamic model of the difference between redundant CPU's due to differences in their clock intervals and uses this model with on-line parameter identification to identify the differences in the clock intervals. The ability of this methodology to accurately track errors due to asynchronisity is demonstrated using a simulated multiprocessor system. The algorithms generate an error signal with the effect of asynchronisity removed and this signal may be used to detect and isolate actual system failures.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Redundant, Asynchronous, Multiprocessor Control, Aeronautics, Parameter Identification	Unclassified-Unlimited Subject Category 08

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	29	

*For sale by the National Technical Information Service, Springfield, Virginia 22161